DRAFT

TACTICAL CONTROL SYSTEM

DATA SERVER
INTERFACE DESIGN DESCRIPTION

VERSION 1.0

April 1997

Prepared by:

NAVAL SURFACE WARFARE CENTER, DAHLGREN DIVISION
Strategic and Strike Systems Department (K63)
Dahlgren, VA 22448

Table of Contents

1. Scope

1.1 Identification

This document provides an Interface Design Description (IDD) between the Data Server (DS) computer software component (CSC) and client application CSCs for the Tactical Control System (TCS). The interface described herein is referred to as the DS Application Program Interface (API). This IDD presents the DS API with complete descriptions and specifications for each procedure and/or function call (i.e., service) available to client applications from the DS.

1.2 System Overview

TBD

1.3 Document Overview

This document is divided into five major sections. The following is a summary of the content of each major section.

Section 1:	Identifies the scope of this document.
Section 2:	Provides a list of all documents referenced or used in the creation of this document.
Section 3:	Provides detailed information on the interface design.
Section 4:	Provides traceability of the interface design to software requirements.
Section 5:	Provides general information that aids in the understanding of the DS and the use of the DS API.

2. Reference Documents

Copies of the specifications, standards, drawings, and publications listed in the next two subsections that are required by suppliers in connection with specified procurement functions should be obtained from the contracting agency or as directed by the contracting officer.

2.1 Military Standards

The following documents listed below form part of this specification to the extent of the organization and format of this document.

DOD-STD-498	Military Standard, Software Development and Documentation
DI-IPSC-81436	Interface Design Description

2.2 TCS Documents

The following documents listed below form part of this specification to the extent of understanding the requirements the DS and its API fulfill to accomplish their task. In the event of conflict between the documents referenced and the content of this specification, the contents of this specification shall be considered a superseding requirement. If the specified revision of the document is not listed, the current approved issue of the document applies.

NSWCDD/96-XX/XXX	TCS Software Design Description
NSWCDD/96-XX/XXX	TCS Software Requirements Specification
NSWCDD/96-XX/XXX	TCS Software Development Plan and Methodology
NSWCDD/97-XX/XXX	Interprocess Communication Interface Design Description

3. Interface Design

3.1 Interface Identification and Diagrams

The Data Sever API consists of three distinct but interrelated pieces:

1. The Interprocess Communication API, which provides the transport and event notification mechanism;
2. The Data Server API itself, which provides the capabillity to create and logically organize data objects; and
3. The Representation and Description Engine (RADE) API, which provides field level access to data that is formated independent of any underlying machine architecture.

The Interprocess Communication package is used directly by the user to open/close a connection to the Data Server as well as to receive notification of completed requests. For information on this package, see the Interprocess Communication IDD. The DS API is used to send service requests to the Data Server via its provided calls. The RADE API is used to access and check individual data fields within data objects handled by the DS API. This chapter describes both the DS API and the RADE API.

This document describes only the interface of a client to the DS. Neither the DS API nor RADE API have a project-unique identifier, but they are generally referred to collectively as the client DS API.

3.2 Data Types

3.2.1 Event Type

Description:

An event is a (symbolically-named) numeric response by the DS to a service request from a client (e.g., create an object). The primary use for events is in conjunction with the callback mechanism for handling DS responses (see Chapter 5 for more information on this mechanism).

The DS reserves 1000 event identifiers (in the range -1000 to -1 inclusive) for this purpose. Currently, it uses only a very small subset of these reserved identifiers. The type `ds_event_typ` defines this subset.

Location:

```
#include <ds_api.h>
```

Specification:

```
typedef enum
{
   null_event          = -1,
   create_event        = -2,
   destroy_event       = -3,
   get_event           = -4,
   set_event           = -5,
   request_key_event   = -6,
   release_key_event   = -7,
   subscribe_event     = -8,
   unsubscribe_event   = -9,
   create_group_event  = -10,
   destroy_group_event = -11,
   enable_group_event  = -12,
   disable_group_event = -13,
   sync_in_progress    = -120,
   sync_done_event     = -121,
        sync_error          = -122,
        build_msg_event     = -200,
        get_group_list      = -201
} ds_event_typ;
```

3.2.2 Status Type

Description:

The status type provides return status values for all DS service requests. It normally signifies the result of the API submitting the request to the DS. It does not represent the final response from the DS for the request. That response is returned via a DS event and the callback mechanism.

Location:

```
#include <ds_api.h>
```

Specification:

```
typedef enum { DS_FAILURE = -1, DS_SUCCESS = 0 } ds_status_typ;
```

3.2.3 Instance Type

Description:

An instance is the result of a create operation for a DS object or group. The instance type provides an identifier for such an instance. It is an opaque type to the user and is used in DS service calls requiring an instance as a parameter or in checking for a valid return from a service which returns an instance (e.g., against the error return value `NULL_INSTANCE`).

Location:

```
#include <ds_api.h>
```

Specification:

```
typedef void *instance_typ;
```

3.2.4 Callback Data Type

Description:

The callback data type contains pertinent DS response information about a service request for an object or a group. It is passed to callback routines associated with DS events. The fields of the callback data type are as follows:

error_code	The response from the DS to the client application regarding the service request. A value of `ERR_NO_ERROR` indicates that the request was successfully completed. The file `interface_error.h` defines the valid values for this field. In Addition each service call enumerates errors that may be returned.
`object`	An instance type identifying the object/group to which the response pertains.
`object_data_ptr`	An indicator of the type of object to which the response pertains. If it is NULL, then the object identifies a group; otherwise, it identifies a data object (See ds_get_data_ptr).
`user_data_ptr`	A pointer to a user-defined block of memory. This is the user_data_ptr parameter originally passed to the ds_register_callback service in associating the callback routine with the DS event.

`msg_ptr`	This field is only pertinent when the callback routine is an Ada procedure. It points to an unconstrained array that is defined as a message type to Ada in the specification file event_msg.ads. **Warning**: Do not access this pointer from a nonAda procedure!

Location:
```
#include <ds_api.h>
```

Specification:
```
typedef struct
{
   int             error_code;
   instance_typ    object;
   void           *object_data_ptr;
   void           *user_data_ptr;
   void           *msg_ptr;
} callback_data_typ;
```

3.2.5 Callback Type

Description:
This type defines the prototype of a callback function to be associated with a DS event. The function has two parameters. The first, *`event`*, is the DS event which caused the function to be called. The second, *`data`*, is a callback data type, as described above, detailing the DS response for this event.

Location:
```
#include <ds_api.h>
```

Specification:
```
typedef void (*callback_typ) ( int event, callback_data_typ *data );
```

3.2.6 Generic Pointer

Description:
This type defines a general pointer which can be used for any type of data structure. Its definition differs depending on whether the user is ANSI C or Classic (K&R) C.

Location:
```
#include <ansi_decls.h>
```

Specification:

ANSI C:	#define PTR void *
Classic C:	#define PTR char *

3.3 Constants

This section describes the general constants used in support of the DS API.

Location:
```
#include <ds_api.h>
```

Specification

ds_standard_group_name	TBD**.**
NULL_INSTANCE	represents an error for those services that return instance_typ.
NO_TRIGGER	represents the value for canceling trigger events.

DS_NODE_NUMBER	TBD**.**

3.4 Error Codes

If an API service request encounters an error in attempting to perform its designated function, it sets an error code for the specific error and returns a general failure value to its caller (e.g., `DS_FAILURE` for a service which returns ds_status_typ or `NULL_INSTANCE` for a service which returns `instance_typ`). The descriptions of the services which follow indicate the specific error codes set by the services. These error codes can be accessed via the `get_errno` service, which is part of the Interface Package.

The DS itself can also return an error code as part of its response to a service request. This code is returned to the client in the callback data structure, which is passed to the callback routine associated with the DS response event. If the client does not have a callback associated with the event, it can retrieve the DS-set error code via the `ds_get_error_code` service.

The `interface_error.h` file defines the valid error codes. See Chapter 5 and the Interface Package IDD for more details.

3.5 Startup and Shutdown Services

3.5.1 Initialization

Description:
The initialization service creates and initializes all the internal data structures for the DS API. It must be called before any of the other API services, to establish a connection with the DS.

Location:
```
#include <ds_api.h>
```

Specification:
```
ds_status_typ ds_init ( void );
```

Parameters:
None

Returns:
```
ds_status_typ
```

DS Response:
NA

Error codes:

`ERR_NO_MEMORY`	Memory could not be allocated for the internal data structures needed for the interface.

3.5.2 DS Reset

Description:
The DS reset service is used to completely initialize the DS. To support “warm” restarts of part or all of TCS, when the DS is first started, it creates all the objects that reside in the persistent data files, thereby allowing it to “'remember”' data objects on behalf of client applications. The reset service clears the persistent data files and destroys all the data objects known to the DS. It does not clear the data object lists that resides in the calling client or in any other connected clients.

This function should only be called by a “master client” immediately after the DS starts executing but before other clients make a connection (as in performing a “cold” start/restart of TCS that is to ignore any previously existing objects).

Location:

```
#include <ds_admin.h>
```

Specification:

```
ds_status_typ ds_master_reset ( int destination );
```

Parameters:

destination	The DS to which to send the master reset message.

Returns:

```
ds_status_typ
```

DS Response:

NA

Error Codes:

ERR_NO_INIT	The interface has not been initialized.
ERR_UNKNOWN_NODE	The destination does not exist.
ERR_CONNECTION_RESET	The connection was closed by a peer during message transmission.
ERR_UNKNOWN_ERROR	A catchall for unanticipated errors.

3.5.3 Client Reset

Description:

The client reset service reinitializes the internal data structures for the DS API (i.e., destroys all objects and groups that have been created). This service could be used as part of error recovery to clear the DS API when it loses contact with the DS. In some cases, a client reset may be preferred to attempting a synchronization (as described in the next section). Optionally, if the DS is still available, this service can send a reset message to the DS requesting that all groups and objects belonging to the client application be destroyed in the DS. In response to this message, the DS will release all the write keys held by the client application, remove the client application from all subscription lists and destroy all objects that have been created by the client application. This reset does not affect other client applications other than another client application may receive a write key as a result of this reset message.

Location:

```
#include <ds_api.h>
```

Specification:

```
ds_status_typ ds_reset ( int destination, int reset_ds );
```

Parameters:

destination	The DS to which to send the client reset message
reset_ds	A switch for sending a message to the DS to destroy all groups and objects belonging to the client application. If it is 0, do not send the client reset message to the DS; otherwise, send it.

Returns:

```
ds_status_typ
```

DS Response:
NA

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.

3.5.4 Synchronization

Description:
The synchronization service assists in error recovery when a DS unexpectedly terminates. Sync is used to transmit all group and object information for a client to a new DS process. It prevents the client from having to reset the API and recreate all of its groups, objects etc.. Sync also resubmits any pending requests on behalf of the client (e.g., a request for a write key). If the ds_sync request returns successfully (i.e., it returns `DS_SUCCESS`), the client **must** wait for an event response, either `sync_done_event` or `sync_error`. `sync_done_event` indicates that the API is done sending all of the group and object information to the DS and the synchronization process is completed. `sync_error` indicates that the DS could not perform an operation which the API requested. During the synchronization process, the client may receive one or more `sync_in_progress` events. These events indicate to the client that the process is proceeding normally.

Location:
```
#include <ds_api.h>
```

Specification:
```
ds_status_typ ds_sync ( int destination );
```

Parameters:

`destination`	The DS to which to send the synchronization messages.

Returns:
```
ds_status_typ
```

DS Response:
```
sync_in_progress
sync_done_event
sync_error
```

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.

3.5.5 Shutdown

Description:
The shutdown service commands the DS to terminate in an orderly manner. It is called by a “master client.” The DS, upon receipt of this message, closes the persistent data files, closes the data log file, closes all client connections and then terminates itself. For a complete orderly shutdown of the DS and any connected client, the 'master client' should inform the peer clients that the DS will be shut down. Each peer client should then close its connection to the DS or should ignore the error message that results from the DS closing the connection to the client.

Location:
```
#include <ds_admin.h>
```

Specification:
```
ds_status_typ ds_shutdown ( int destination );
```

Parameters:

destination	The DS to which to send the shutdown message.

Returns:
```
ds_status_typ
```

DS Response:
NA

Error Codes:

ERR_NO_INIT	The interface has not been initialized.
ERR_UNKNOWN_NODE	The destination does not exist.
ERR_CONNECTION_RESET	The connection was closed by a peer during message transmission.
ERR_UNKNOWN_ERROR	A catchall for unanticipated errors.

3.6 Group Services

Groups provide a mechanism for associating data objects that have some common characteristic (e.g., represent a particular Air Vehicle for a particular mission). The objects in the group are the *children* of the group, and the group is the *parent* of the objects it contains. **Note**: Objects can be arranged in parent-child trees themselves, so the actual direct parent of a given object may be another object; however, the top-level parent of that object will be the group which contains the root object of the parent-child tree containing the object.

The *standard group* is a predefined group in the DS. It is available for containing objects that do not "fit" into any user-defined group.

3.6.1 Create Group

Description:
The create group service creates a group with a given name for the client and registers that group with the DS.

Location:
```
#include <ds_api.h>
```

Specification:
```
instance_typ ds_create_group ( int     destination,
                               char   *group_name,
                               int     enable );
```

Parameters:

destination	The DS to which to send the message for registering the group.
group_name	The name of the group to create.
enable	A switch for having the group enabled or disabled when created. If it is 0, the group is disabled on creation; otherwise, the group is enabled on creation.

Returns:
```
instance_typ
```

DS Response:

`create_group_event`

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_NO_MEMORY`	Memory could not be allocated to create the group.
`ERR_PARAMETER`	A group with the given name already exists for this client.

3.6.2 Destroy Group

Description:

The destroy group service does the following:

1. Destroys the group and any child objects of the group for the DS API, and
2. Sends a message to the DS requesting that the group be destroyed.

The DS sends only one response for this request, i.e., for the destroy action on the group itself. It does not send responses for any child objects that it destroys as a result of destroying the group.

Location:

`#include <ds_api.h>`

Specification:

`ds_status_typ ds_destroy_group ( instance_typ group );`

Parameters:

`group`	An identifier for a group.

Returns:

`ds_status_typ`

DS Response:

`destroy_group_event`

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_PARAMETER`	If *`group`* is the standard group; or if there is a pending create or destroy response for *`group`* (i.e., a create/destroy message has been sent to the DS but the DS has not replied with the corresponding event yet.)
`ERR_UNKNOWN_GROUP`	*`group`* does not reference valid group.

3.6.3 Enable Group

Description:

The enable group service enables the receipt of subscriptions from objects that are members of the selected group. Immediately upon receipt of the enable group request, the DS sends the issuing client all objects from the group that have changed while the group has been disabled.

Location:

```
#include <ds_api.h>
```

Specification:

```
ds_status_typ ds_enable_group ( instance_typ group );
```

Parameters:

group	An identifier for a group.

Returns:

```
ds_status_typ
```

DS Response:

`enable_group_event`	
`subscribe_event`	If the DS sends any outstanding subscriptions as a result of a group being enabled, the client will receive a subscription event for each outstanding subscription.

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_UNKNOWN_GROUP`	The group does not reference a valid group.
`ERR_PARAMETER`	The group has a pending create or destroy response (i.e., a create/destroy message has been sent to the DS but the DS has not replied with the corresponding event yet.)

3.6.4 Disable Group

Description:

The disable group service disables the receipt of subscriptions from objects that are members of the group identified by the group parameter. Immediately upon receipt of the disable group request, the DS marks the group as inactive for the client and stop sending the client subscriptions for objects that belong to the group. If any of the group objects that the client has subscribed to change while the group is disabled, the DS marks those objects as changed for subsequent forwarding to the client if the group is re-enabled at a later time.

Location:

```
#include <ds_api.h>
```

Specification:

```
ds_status_typ ds_disable_group ( instance_typ group );
```

Parameters:

group	An identifier for a group.

Returns:

```
ds_status_typ
```

DS Response:

```
disable_group_event
```

Error Codes:

`ERR_NO _INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.

`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_UNKNOWN_GROUP`	The group does not reference a valid group.
`ERR_PARAMETER`	The group has a pending create or destroy response (i.e., a create/destroy message has been sent to the DS but the DS has not replied with the corresponding event yet.)

3.6.5 Standard Group

Description:
The standard group service returns the instance identifier for the predefined standard group.

Location:
```
#include <ds_api.h>
```

Specification:
```
instance_typ ds_standard_group ( int destination );
```

Parameters:

`destination`	The DS for which to retrieve the standard group identifier.

Returns:
```
instance_typ
```

DS Response:
NA

Error Codes:

`ERR_NO _INIT`	The interface has not been initialized.
ERR_PARAMETER	

3.6.6 Find Object By Name

Description:
The find object by name service returns the instance of a data object with the given name if that object is contained in the given group.

Location:
```
#include <ds_api.h>
```

Specification:
```
instance_typ ds_find_object_by_name ( instance_typ    group,
                                      const char     *object_name );
```

Parameters:

`group`	The group to search for the object.
`object_name`	The name of the object to find.

Returns:
```
instance_typ
```

DS Response:
NA

Error Codes:

`ERR_PARAMETER`	If *`group`* does not reference a valid group.

3.6.7 Find Group By Name

Description:

The find group by name service returns the instance of the group with the given name in the designated DS.

Location:

```
#include <ds_api.h>
```

Specification:

```
instance_typ ds_find_group_by_name ( int           destination,
                                     const char   *group_name );
```

Parameters:

destination	The DS in which to find the group.
group_name	The name of the group to find.

Returns:

```
instance_typ
```

DS Response:

NA

Error Codes:

NA

3.7 Object Services

From the DS API point of view, an object is a collection of data that have meaning as a unit (e.g., flight status for an Air Vehicle). Each object created by a client **must** belong to a group, although different instances of the same object can be defined in different groups. Objects can also be arranged in hierarchical trees, with one object serving as parent for one or more child objects on the next level away from the root object of the tree.

The DS itself does not care about the data within an object. To the DS, the objects are simply buckets of specified sizes. These buckets can be defined either by specifying the size of the bucket or by referencing a data-object schema. The schema contains all of the information necessary for creating the object and populating it with default data. It also provides a means of accessing individual datum fields within the object via DS API services (see section 3.**X**).

Note: These services apply only to objects, i.e., they cannot be applied to an instance of a group. Groups are collections of objects and do not contain data themselves, so it does not make sense to perform operations such as getting a write key, setting, getting, etc., on them.

3.7.1 Create Object

Description:

The create object service provides an interface for creating a data object. It accepts either a size for the object or an option specifying the object to be a RADE object. If the requester designates the object as a RADE object, the service ignores the size parameter. In this case, all of the information need for object creation comes from the object schema, which the DS can access via the object name. The DS also makes the schema available to the DS API for use by the data accessing functions.

The create object service creates an object as follows:

1. Ensures that the size parameter > 0 (unless this is a RADE_OBJECT create),
2. Verifies that the parent has been created and has received a create response from the DS,

3. Verifies that an object by the same name does not exist in the group to which the parent belongs,
4. Allocates memory for the object, and
5. Sends a message to the DS requesting the creation of the object.

Location:
```
#include <ds_api.h>
```

Specification:
```
   instance_typ ds_create_object ( int             destination,
                                   instance_typ    parent,
                                   char           *object_name,
                                   short           size,
                                   int             options );
```

Parameters:

destination	The DS to which to send the message.
parent	The parent of the object being created; this may be either a group or an object.
object_name	The name of the object to create.
size	The size of the object to be created.
options	A bit field constructed by ORing option values from the following list:

NO_OP	This should be used in the absence of any other option.
PERSISTENT_OP	This object attribute indicates to the DS that when a client 'sets' the object a copy of the object is written to the persistent storage file.
LOG_DATA_OP	This object attribute indicates to the DS that when a client 'sets' the object a copy of the object is written to the data log file.
WRITE_KEY_OP	This is a request for the write key. This eliminates the need for a separate write key API call. When the client receives the key, the DS will respond to it with a request_write_key event. Alternatively, when the client receives the create response from the DS, it can call ds_received_write_key to determine if it was successful in obtaining the key.

`SUBSCRIBE_OP`	This is a request by the client to be added to the subscription list for the object. This eliminates the need for a separate call to the subscribe API call. If there is valid data for the object in the DS, the DS will send a copy of the object to the client and respond to the client with a subscribe_event. Also, when the client receives the create response from the DS, it can call ds_received_valid_data to determine if the DS sent object data along with the create response. If ds_received_valid_data returns true, then the clients copy of the data is current with the copy held by the DS.
`GET_DATA_OP`	This is a request by the client to get a copy of the object data if the copy held by the is valid (i.e., another client has 'set' the object). This eliminates the need for a separate call to the get API call. If the DS sends a copy of the object to the client, it also sends a get_event along with it. Also, when the client receives the create response from the DS, it can call ds_received_valid_data to determine if the DS sent object data along with the create response. If ds_received_valid_data returns true, the clients copy of the data is current with the copy held by the DS. This option is especially useful for retrieving data for an object that was created with the PERSISTENT_OP attribute during a previous DS execution.
`RADE_OBJECT`	This is a request to the DS to create the object via a schema. The size parameter has no meaning when this option is used.

<u>Returns</u>:
`instance_typ`

<u>DS Response</u>:

`create_event`	
`request_key_event`	When the client receives the write key, if the create included option `WRITE_KEY_OP`.
`Subscribe_event`	When the client receives an initial copy of the object, if the create included option `SUBSCRIBE_OP`.
`Get_event`	When the client receives a valid copy of the object, if the create included option `GET_DATA_OP`.

<u>Error Codes</u>:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_PARAMETER`	*`size`* is invalid, i.e., greater than `MAXSHORT` as defined in `<values.h>.`

`ERR_DUPLICATE`	The object already has been created as a child of parent.
`ERR_NO_MEMORY`	Memory could not be allocated for the object.
`ERR_NO_RESPONSE`	The client has not received the response from the service call to create parent.

3.7.2 Destroy Object

Description:

The destroy object service destroys an object and its children as follows:

1. Verifies that the object is valid (has been created),
2. Verifies that the object does not have a pending destroy_event (i.e., the client has not already requested the object be destroyed, either directly or as a child of another object),
3. Verifies that the object does not have a pending create_event,
4. Sends a destroy request to the DS, and
5. Marks the object and its children as destroyed.

The object and its children are not actually destroyed in the DS API until the client receives the destroy response. When this happens, the API performs a recursive destroy of all of these objects.

Location:
```
#include <ds_api.h>
```

Specification:
```
ds_status_typ ds_destroy_object ( instance_typ object );
```

Parameters:

`object`	The instance of an object to destroy.

Returns:
```
ds_status_typ
```

DS Response:

`destroy_event`	One event for each object actually destroyed by the service (the input object and each of its children).

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_PARAMETER`	The object has a pending create or destroy response (the destroy response can be either from itself or its parent).
`ERR_UNKNOWN_OBJECT`	The object has not been created.

3.7.3 Set Object

Description:

The set object service sends the client copy of a data object to the DS, provided that the client holds the write key for the data object.

Location:
```
#include <ds_api.h>
```

Specification:
```
ds_status_typ ds_set ( instance_typ   object,
                       int            respond );
```

Parameters:

`object`	The instance of an object to set.
`respond`	If nonzero, the DS is to respond to the client upon accomplishing the set.

Returns:

```
ds_status_typ
```

DS Response:

`set_event`	If the client has asked the DS to respond to this service request.

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_UNKNOWN_OBJECT`	The object has not been created.
`ERR_NO_RESPONSE`	The object has a pending create or destroy response (the destroy response can be either from itself or its parent).
`ERR_WRITE_KEY_NOT_HELD`	The client does not hold the write key for the data object.

3.7.4 Get Object

Description:

The get object service requests that the DS send its copy of the data object to the client. The DS will fulfill this request only if the data object has been 'set'.

Location:

```
#include <ds_api.h>
```

Specification:

```
ds_status_typ ds_get ( instance_typ object );
```

Parameters:

`object`	The instance of an object to get.

Returns:

```
ds_status_typ
```

DS Response:

```
get_event
```

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_NO_RESPONSE`	The object has a pending create or destroy response (the destroy response can be either from itself or its parent).
`ERR_UNKNOWN_OBJECT`	The object has not been created.

3.7.5 Request Key for Object

Description:

To be able to set the client copy of an object instance into the DS, the client must hold the write key for that object instance. There are two way for the client to get this key, by requesting it as an option when creating the object instance or via this service.

The request key for object service requests the write key for a data object from the DS. The DS will fulfill the request when the key is available (if not immediately). If the key is not available, the client is put on a waiting list for the key. When the key is released by the holding client, the next client on the waiting list will be sent the key. If the client already holds the key for a data object, it is not an error for the client to request the key again. In this case, the DS will simply send a success response.

Location:

```
#include <ds_api.h>
```

Specification:

```
ds_status_type ds_request_key ( instance_typ object );
```

Parameters:

object	The instance of an object for which the client is requesting the write key.

Returns:

```
ds_status_typ
```

DS Response:

```
request_key_event
```

Error Codes:

ERR_NO_INIT	The interface has not been initialized.
ERR_UNKNOWN_NODE	The destination does not exist.
ERR_CONNECTION_RESET	The connection was closed by a peer during message transmission.
ERR_UNKNOWN_ERROR	A catchall for unanticipated errors.
ERR_NO_RESPONSE	The object has a pending create or destroy response (the destroy response can be either from itself or its parent).
ERR_UNKNOWN_OBJECT	The object has not been created.

3.7.6 Release Key for Object

Description:

The release key for object service requests the DS to release the client's write key for a data object and make it available for other clients to use (i.e., allow another client to set the data object).

Location:

```
#include <ds_api.h>
```

Specification:

```
ds_status_typ ds_release_key ( instance_typ   object,
                               int            respond );
```

Parameters:

object	The instance of an object for which the DS is to release the write key.
respond	If nonzero, the DS is to respond to the client upon releasing the key.

Returns:

```
ds_status_typ
```

DS Response:

`release_key_event`	If the client has asked the DS to respond to this service request.

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_NO_RESPONSE`	The object has a pending create or destroy response (the destroy response can be either from itself or its parent).
`ERR_UNKNOWN_OBJECT`	The object has not been created.

3.7.7 Subscribe to Object

Description:

The subscribe to object service requests that the DS add the client to the subscription list for the object. Then, in response to a set request for the data object, the DS will send each client on the subscription list a current copy of the data object. Also, in response to the subscription request, the DS will send the requesting client a copy of the data object, if the data object contains valid data (i.e., if a client has set the object).

Location:

```
#include <ds_api.h>
```

Specification:

```
ds_status_typ ds_subscribe ( instance_typ object );
```

Parameters:

`object`	The instance of an object to which the client wishes to subscribe.

Returns:

```
ds_status_typ
```

DS Response:

`subscribe_event`	Each time the DS sends a copy of the data object to the client.

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_NO_RESPONSE`	The object has a pending create or destroy response (the destroy response can be either from itself or its parent).
`ERR_UNKNOWN_OBJECT`	The object has not been created.

3.7.8 Unsubscribe to Object

Description:

The unsubscribe to object service requests that the DS remove the client from the subscription list for the object.

Location:

```
#include <ds_api.h>
```

Specification:

```
ds_status_typ ds_unsubscribe ( instance_typ   object,
                               int            respond );
```

Parameters:

object	The instance of an object to subscribe to.
respond	If nonzero, the DS is to respond to the client upon removing the object subscription.

Returns:

`ds_status_typ`

DS Response:

`unsubscribe_event`	If the client has asked the DS to respond to this service request.

Error Codes:

`ERR_NO_INIT`	The interface has not been initialized.
`ERR_UNKNOWN_NODE`	The destination does not exist.
`ERR_CONNECTION_RESET`	The connection was closed by a peer during message transmission.
`ERR_UNKNOWN_ERROR`	A catchall for unanticipated errors.
`ERR_NO_RESPONSE`	The object has a pending create or destroy response (the destroy response can be either from itself or its parent).
ERR_UNKNOWN_OBJECT	The object has not been created.

3.7.9 Get Object Group

Description:

The get object group service returns the identifier for the group to which the input object belongs.

Location:

```
#include <ds_api.h>
```

Specification:

```
instance_typ ds_get_group ( instance_typ object );
```

Parameters:

object	The instance of an object for which to get the group.

Returns:

The group to which the object belongs if the object is valid; otherwise, `NULL_INSTANCE.`

DS Response:

NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object has not been created.
`ERR_NO_ERROR`	The object is a valid object.

3.7.10 Get Object Parent

Description:

The get object parent service returns the instance identifier for the parent of the input object. This parent may be either a group or another object.

Location:

```
#include <ds_api.h>
```

Specification:

```
instance_typ ds_get_parent ( instance_typ object );
```

Parameters:

`object`	The instance of an object for which to get the parent.

Returns:

The parent of the object if object is a valid identifier; otherwise, `NULL_INSTANCE`.

DS Response:

NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object has not been created.
`ERR_NO_ERROR`	The object is a valid object.

3.7.11 Get Object Data Pointer

Description:

The get object data pointer service returns a pointer to the data associated with the object. This service produces a useful result only if the client created the object directly, not via a schema. In other words, this service does not return any client-usable information for an object created with the `RADE_OBJECT` option. It does not set any error code to indicate this condition; rather, it is the client's responsibility to know which objects it created directly and which it created as `RADE_OBJECT`s.

Location:

```
#include <ds_api.h>
```

Specification:

```
PTR ds_get_data_ptr ( instance_typ object );
```

Parameters:

object	the instance of an object for which to get the data pointer.

Returns:

A pointer to the data associated with the object, if object is a valid identifier; otherwise, `NULL`

Note: The pointer returned for an object created with the `RADE_OBJECT` option does not access the actual object data, so it is of no use to a client!

DS Response:

NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object has not been created.
`ERR_NO_ERROR`	The object is a valid object.

3.7.12 Get Object Size

Description:

The get object size service returns the size of the data block associated with the object. This service produces a useful result only if the client created the object directly, not via a schema. In other words, this service does not return any client-usable information for an object created with the

`RADE_OBJECT` option. It does not set any error code to indicate this condition; rather, it is the client's responsibility to know which objects it created directly and which it created as `RADE_OBJECT`s.

Location:

```
#include <ds_api.h>
```

Specification:

```
int ds_get_object_size ( instance_typ object );
```

Parameters:

`object`	The instance of an object for which to get the data block size.

Returns:

The size of the object if object is a valid identifier; otherwise, -1.
Note: The size returned for an object created with the RADE_OBJECT option includes more than just the object data block, so it is of no use to a client!

DS Response:

NA

Error Codes:

None

3.7.13 Object Received Valid Data

Description:

The object received valid data service allows the client to determine if the local object has received data from the DS at any time during its existence (i.e., from a subscription, an explicit `ds_get` request, or the `ds_create_object` `GET_DATA_OP` option).

Location:

```
#include <ds_api.h>
```

Specification:

```
int ds_received_valid_data ( instance_typ object );
```

Parameters:

`object`	The instance of an object to query.

Returns:

A nonzero value if object is a valid identifier and the local object has received data from the DS; otherwise, 0.

DS Response:

NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object has not been created.
`ERR_NO_ERROR`	The object is a valid object.

3.7.14 Object Received Write Key

Description:

The object received write key service allows the client to determine if the local object has received the write key from the DS (either from an explicit `ds_request_key` request or from the `ds_create_object` `WRITE_KEY_OP` option).

Location:

```
#include <ds_api.h>
```

Specification:

```
int ds_received_write_key ( instance_typ object );
```

Parameters:

object	The instance of an object to query.

Returns:

A nonzero value if object is a valid identifier and the local object has received the write key from the DS; otherwise, 0.

DS Response:

NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object has not been created.
`ERR_NO_ERROR`	The object is a valid object.

3.8 Common Services

3.8.1 Register Callback

Description:

The register callback service associates a client function with a DS event for an object or group. (See Chapter 5 for more information on the use of callbacks and the event handling mechanism.) If the client does not wish to provide a function, it should use one of the two predefined functions default_func or `ignore_func`. The `default_func` is a directive to the DS API to pass event notification to the client through the normal event notification mechanism. The `ignore_func` is a directive to the DS API not to pass the event notification to the client. For both the `default_func` and the `ignore_func` the API does all processing for the event as normal, including the functionality of the set trigger API call.

Location:

```
#include <ds_api.h>
```

Specification:

```
ds_status_typ ds_register_callback ( instance_typ   instance,
                                     ds_event_typ   event,
                                     PTR            user_data_ptr,
                                     callback_typ   routine );
```

Parameters:

instance	An object or group for which to register a callback
event	One of the event identifiers (see Section 3.2.1)
user_data_ptr	A pointer to a user-defined block of data

`routine`	A pointer to a function to be called when the event arrives from the DS. If the client does not wish to provide a function, it should use one of the two predefined functions:

`default_func`	This function parameter is a directive to the DS API to pass the event notification to the client through the normal event handling mechanism.
`ignore_func`	This function parameter is a directive to the DS API not to pass the event notification to the client.

Returns:

`ds_status_typ`

DS Response:

NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object or group has not been created.
`ERR_PARAMETER`	If event is not one of the event identifiers.

3.8.2 Get Name

Description:

The get name service returns a pointer to the name corresponding to the input object/group instance identifier.

Location:

```
#include <ds_api.h>
```

Specification:

```
const char *ds_get_name ( instance_typ instance );
```

Parameters:

`instance`	An object or group for which to get the name.

Returns:

A pointer to the name of the object/group if instance is a valid identifier; otherwise, `NULL.`

DS Response:

NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object or group has not been created.

3.8.3 Get Error Code

Description:

The get error code service returns the error code that has been returned most recently from the DS regarding the object/group referenced. It is primarily for use by client that is has not associated a callback routine with one or more DS events.

Location:

```
#include <ds_api.h>
```

Specification:
```
int ds_get_error_code ( instance_typ instance );
```

Parameters:

instance	An object or group for which to get the error code.

Returns:
An error code if instance is a valid identifier; otherwise, -1.

DS Response:
NA

Error Codes:
None

3.8.4 Set User Data

Description:
The set user data service allows a client to associate data with an instance of an object or group. The service does not require any particular form for the data. Its form and use are strictly client-determined. **Note**: One set of user data is available for setting for each object or group. This is not the same user data associated with an event-specific callback for an object or group. Each of these callbacks can have a different user data item (specified as *user_data_ptr* in the `ds_register_callback` service call).

Location:
```
#include <ds_api.h>
```

Specification:
```
int ds_set_user_data ( instance_typ   instance,
                       PTR            data );
```

Parameters:

instance	An object or group for which to set the user data.
data	The user data to associate with the instance.

Returns:
0 if the data is successfully associated with the instance; -1 otherwise.

DS Response:
NA

Error Codes:

ERR_UNKNOWN_OBJECT	The object or group has not been created
ERR_NO_ERROR	The data is successfully associated with the object or group.

3.8.5 Get User Data

Description:
The get user data service allows a client to retrieve data that was previously associated with an instance of an object or group by the set user data service.

Location:
```
#include <ds_api.h>
```

Specification:

```
PTR ds_get_user_data ( instance_typ instance );
```

Parameters:

instance	An object or group for which to get the user data.

Returns:

A pointer to the user data if instance is a valid identifier; otherwise, `NULL`.

DS Response:

NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object or group has not been created.
`ERR_NO_ERROR`	The data pointer is successfully returned.

3.8.6 Set Trigger Event

Description:

The set trigger service provides a mechanism for associating a DS event with an Object Manager event. **Warning:** This service is provided for Ada clients only. It should not be called from clients written in C.

Location:

#include <ds_api.h>

Specification:

```
ds_status_typ ds_set_trigger_event ( instance_typ   instance,
                                     ds_event_typ   event,
                                     int            code);
```

Parameters:

instance	An instance of an object or group.
event	One of the event identifiers (see Section 3.2.1).
code	The code to generate for the Object Manager when the event occurs.

Returns:

`ds_status_typ`

DS Response:

NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object or group has not been created.
`ERR_PARAMETER`	If event is not one of the event identifiers.
`ERR_NO_ERROR`	If the trigger code set is successful.

3.8.7 Get Trigger Event

Description:

The get trigger service returns the external event that is generated when the DS event for the object/group referenced arrives. **Warning**: This service is provided for Ada clients only. It should not be called from clients written in C.

Location:
```
#include <ds_api.h>
```

Specification:
```
   int ds_get_trigger_event ( instance_typ   instance,
                              ds_event_typ   event );
```

Parameters:

instance	An instance of an object or group.
event	One of the event identifiers.

Returns:
An event identifier (i.e., the code for the set trigger event).

DS Response:
NA

Error Codes:

`ERR_UNKNOWN_OBJECT`	The object or group has not been created.
`ERR_PARAMETER`	If event is not one of the event identifiers.
`ERR_NO_ERROR`	If a trigger code is returned (including `NO_TRIGGER`).

3.9 Miscellaneous Services

3.9.1 Get Recent Group

Description:
The get recent group service returns the group instance that has received the most recently delivered group-related event from the DS.

Location:
```
#include <ds_api.h>
```

Specification:
```
instance_typ ds_get_recent_group ( void );
```

Parameters:
None

Returns:
The group from the most recent DS group-related event; NULL_INSTANCE if the API has not yet received a group-related DS event.

DS Response:
NA

Error Codes:
None

3.9.2 Get Recent Object

Description:
The get recent object service returns the object instance that has received the most recently delivered object-related event from the DS.

Location:
```
#include <ds_api.h>
```

Specification:
```
instance_typ ds_get_recent_object ( void );
```

Parameters:
None

Returns:
The object from the most recent DS object-related event; `NULL_INSTANCE` if the API has not yet received an object-related DS event.

DS Response:
NA

Error Codes:
None

3.10 Representation and Description Engine (RADE)

TBD

4. Requirements Traceability

This section has been tailored out.

5. Notes

The sections of this chapter provide background information for understanding the general operation of the DS, the use of the API, and the relationship of the DS API to other APIs which support DS use.

5.1 Overview

5.1.1 General

The primary function of the DS is to provide a mechanism for the distribution of data between clients. The DS is implemented as a client-server architecture that provides a central repository of data for a distributed system. It provides functions to allow a client to send and receive data, store data for retrieval during a subsequent execution (persistent storage) and log data (for later analysis). It also provides the developer with debugging capability from the unit level through system level.

The DS supports a very small command set for data manipulation. This command set allows clients to create/destroy data object, read/write data objects, and subscribe to changes in data objects. It also provides for grouping of related data objects. Data objects in the API and DS do not have structure; they are simply block of data. The DS sends and receives these blocks of data without the need to know their content. It is the client that gives meaning to the contents of data objects.

Persistent storage is a mechanism by which the DS and clients can recover from an unexpected system shutdown (i.e., a client process or DS terminates abnormally). When the client creates data objects, it can set the persistence attribute. This attribute informs the DS that any changes (writes) to the data object should also be written to a persistent storage file. For example, a client could create a data object that contains information describing the state in which the client is executing. If for some unforeseen reason the client terminates, upon restart, it could reconnect to the DS and retrieve this “state” object to determine its state prior to the unexpected termination. This method is not foolproof, but it does provide a mechanism by which clients can recover from system faults with a reasonable degree of certainty.

Data logging is a mechanism clients can use to record data for analysis during/after execution. The log is a file containing a time ordered set of selected objects. Whenever any of the selected objects is set into the DS (via a ds_set service request), that object is time-stamped and written into the log file. A client can select an object for logging by setting the LOG_DATA_OP when it creates the object.

5.1.2 Groups

TBD

5.1.3 Objects

TBD

5.1.4 Normal Event Handling

TBD

5.2 Theory of Operation

TBD

5.2.1 The Event Model

TBD

5.2.2 Object Writer

TBD

5.2.3 Object Reader

TBD

5.2.4 Object Subscriber

TBD

5.3 The Subscription (Automatic Data Forwarding)

TBD

6. Acronyms

API	Application Program Interface
CSC	Computer Software Component
DS	Data Server
IDD	Interface Design Description
RADE	Representation and Description Engine
TCS	Tactical Control System